Preprints of the 2012 20th Mediterranean Conference on Control & Automation (MED), Barcelona, Spain, July 3-6, 2012

Generating Safe and Equally Long Trajectories for Multiple Unmanned Agents

Ulrik Jørgensen
Department of Marine Technology
Norwegian University of Science and Technology
7491 Trondheim, Norway
Email: ulrik.jorgensen@ntnu.no

Roger Skjetne
Department of Marine Technology
Norwegian University of Science and Technology
7491 Trondheim, Norway
Email: skjetne@ieee.org

***Abstract*— In this paper a path planning method for multiple unmanned agents is presented. The proposed algorithm utilizes Dubins paths such that the final paths will be feasible for agents with a given turning constraint. The algorithm further ensures that the agents will avoid collisions. For cases where it is important to arrive simultaneously, this is achieved by assuming that the vehicles are operating with constant speeds and then create equally long paths. The main challenge is, however, to ensure that the algorithm is computationally efficient, as it is intended for small sized unmanned vehicles where decisions have to be taken in a short time and calculated with restricted computational units.**

The proposed algorithm is tested with a case study that illustrates the findings.

***Index Terms*— Path planning, autonomous agents, collision avoidance, safety.**

I. Introduction

Autonomous systems working without human intervention are a well known research field and are investigated thoroughly. A reason for this is that unmanned systems are perfectly suited for handling hazardous material, operate in dangerous areas, and perform time-consuming and repetitive missions. In order to increase the efficiency compared to single vehicle operations; academic and commercial research have looked into the field of using systems consisting of multiple autonomous agents. Increasing the numbers of co-operating agents will in addition to the mentioned advantages have the opportunity to decrease operational time, increase the cost-efficiency, and make the system fault-tolerant.

In order to achieve the goal of the system, the autonomous system needs to be capable of making intelligent decisions on their own. For unmanned agents working in their given environment, one such intelligent decision would be to generate optimal paths to follow. An example of this can be seen in Figure 1. The path for an agent needs to be feasible, meaning that the agent with its given constraints, is capable of following the path. Moreover, when a vehicle is going from a point to a target point, it is necessary to plan a path that avoids obstacles in the environment, while still being feasible.

In military operations, it is sometimes desired for all agents to arrive simultaneously. This is for instance the case for military attacks where a simultaneous arrival will decrease the likelihood of being detected as well as increasing the element of surprise. Another application that benefits from a coordinated rendezvous is when information is going to be exchanged between multiple agents. In many cases, for instance underwater operations, the communication range is limited and the vehicles must therefore be in close distance such that the needed data can be transmitted among each other.

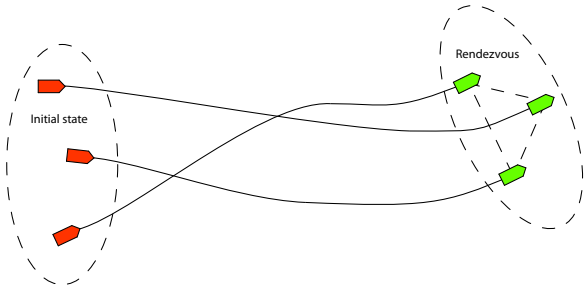

Fig. 1. Three agents in an initial state, planning to rendezvous for some operational reason.

Unmanned vehicles are often quite small in size and have a limited computational unit. This is further complicated in operations where the speed of the vehicles is high and decisions have to be taken in a short time. As a result, this leads to hard constraints for the path planning algorithm as it must calculate the paths in a short time with limited CPU capability. It is therefore a prerequisite of this paper that the computational capabilities of the agents are limited, and, that the path planning procedure must be computationally effective.

A lot of research has been done on the topic of path planning for autonomous systems. Most of this work is, however, done in the field of robotics [1]. Nevertheless, there exists a considerable number of papers written on the topic of path planning for vehicles where especially unmanned aerial vehicles (UAVs) are treated. Several path planning techniques have been developed using for instance probability maps [2], [3], dynamic programming [4], mixed-integer linear programming (MILP) [5], road map methods [6], and potential field methods [7]. Another path generation

Research supported by the RCN project 199567: “Arctic DP”.

978-1-4673-2529-5/12/$31.00 ©2012 IEEE

technique is presented in [8, Chapter 2], where a smooth path continuously parametrized by a scalar variable is generated from the specification of waypoints. See also [9] for further results on maneuvering control designs.

None of these papers consider the aspect of arriving at the same time. This is, however, treated in [10] and [11], where simultaneous arrival is solved by using a chain analogy where segments are added or subtracted in order to change the length of the path. Assuming that the UAVs are flying with a constant speed, it is therefore possible for all the vehicles to arrive at the same time. Another similar method is presented in [12] that has inspired this work.

The generated path for each agent has to satisfy certain requirements. One of these requirements is the minimum turning radius associated with nonholonomic vehicles. Several methods are developed to handle this constraint, for instance the "Clothoid paths" [13], [14] or the "Pythagorean hodographs" [15]. These two methods generate smooth paths where the curvature is continuous. However, as no analytic solution is available, they are in general computationally demanding. Consequently, they rely on an unknown number of iterations, which is unpractical in real-time systems where fixed execution time is preferable. Another solution for constructing feasible paths is the commonly used "Dubins paths" [16]. The main reason for its popularity is that the Dubins paths are easy to implement since there exist exact analytic expressions for the paths. Moreover, the Dubins paths are proven to be the shortest possible path for vehicles that have a minimum turning radius. This is important since reducing the travel distance will reduce the fuel consumption and minimize the travel time. Nevertheless, the relaxed computational demands are most important for this work and chosen to generate the feasible paths.

II. Problem formulation

We consider the problem of generating paths for multiple unmanned agents, where all the agents are operating in a 2D space. The main objective of the path planning algorithm is to ensure that all the produced paths are (i) feasible, (ii) safe, and (iii) arrive at the same time.

III. Feasible, safe, and equally long paths

A. Main algorithm

In order to make the paths feasible, the minimum turning radius $\rho \geq \rho_{\min}$ for each path and its intended agent must be satisfied. This is achieved by using the Dubins path method. Once the paths are made feasible, the next important step is to ensure that the paths are safe. Being safe means that there will be no collision as long as all the agents follow their paths. This is the most crucial requirement for the path planner, and it should always be satisfied. However, being safe does not mean that paths cannot cross each other. Crossing between two paths is accepted as long as the crossing does not occur at the same time and with a reasonable safety margin.

The third and final requirement for the generated paths is that all agents are going to arrive at their targets at the same time. Assuming that the vehicles are operating with a constant speed, this implies that all the paths must be equally long to ensure simultaneous arrival.

In order to satisfy all the requirements, Algorithm 1 is proposed. The "start pose" and "finish pose" is denoted P_{si} and P_{fi}, respectively. R_i is the safety radius, $r_i(q)$ is the final calculated path, and the subscript i denotes the i-th agent. The three steps of the main algorithm will be carefully explained in the following sections.

Algorithm 1: Main algorithm.
function MAKEPATHS(P_{si}, P_{fi}, R_i) ▷ $i \in \{1, \cdots, n\}$
  Create feasible paths.
  while not safe & not equally long **do**
    Create safe paths.
    Create equally long paths.
  end while
  return $r_i(q)$ ▷ Feasible, safe, and equally long paths.
end function

B. Creating feasible paths

Assume that the system consists of n agents, where all the vehicles are operating in a 2D space. This restriction can be expanded to a three-dimensional case, but for simplicity of presentation, two dimensions are chosen.

The main goal of the path algorithm is to connect a start position (x_{si}, y_{si}) with heading ϕ_{si}, with the desired and final position (x_{fi}, y_{fi}), with heading ϕ_{fi}. The subscript s denotes "start", f denotes "finish", while i denotes the i-th agent where $i \in \{1, \cdots, n\}$. The start pose $P_{si} = P_{si}(x_{si}, y_{si}, \phi_{si})$ and the finish pose $P_{fi} = P_{fi}(x_{fi}, y_{fi}, \phi_{fi})$. This notation will be utilized throughout the paper and is in consistency with [12].

Dubins paths are in general constructed by connecting two circular arcs by a straight segment that is tangential to the arcs. In practice there exist four possible ways to connect the start pose to the finish pose by using two arc segments and one straight segment, i.e. $\{LSL, RSL, LSR, RSR\}$, where R, L, and S denotes right-turning arcs, left-turning arcs, and straight segments, respectively. The four possible Dubins paths are illustrated in Figure 2. A straight forward algorithm for calculating these paths is presented in [12] where the shortest of the four paths is used[1].

C. Creating safe paths

A real operating environment is often filled with obstacles and areas to avoid, and this must be handled carefully by the path planner. Assuming that the obstacles are known in advance, it is possible to modify the generated path by changing the start radius ρ_{si} and the finish radius ρ_{fi} in such a way that the obstacles are avoided while still ensuring that ρ_{si} and $\rho_{fi} \geq \rho_{\min,i}$. This will be the strategy in order to make the paths safe and equally long. An illustration can be seen in Figure 3.

[1]Note that some errors are present in the calculations of the Dubins paths shown in [12].

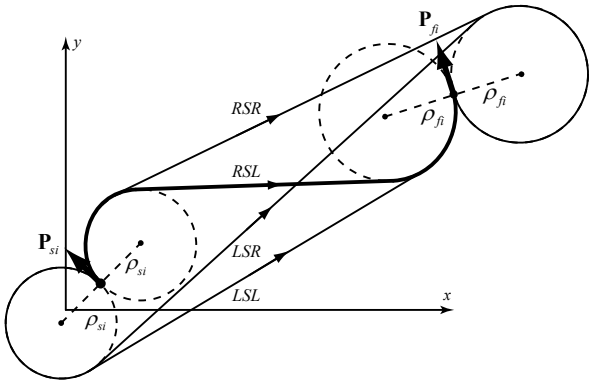

Fig. 2. The four possible Dubins paths where the shortest path is illustrated with bold lines.

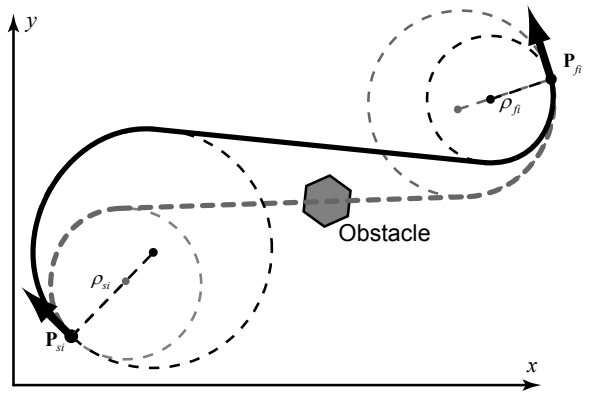

Fig. 3. Dubins paths connecting the start pose P_{si} with the finish pose P_{fi}. The initial (gray) path goes through an obstacle, and it is therefore created another modified path (black). The turning radius ρ_{si} and ρ_{fi} must in both cases be larger than the minimum turning radius of the agent $\rho_{\min,i}$.

For simplicity of presentation we assume that there are no static obstacles in the operational environment, and that the only possibility for collision is if the agents collide with each other. This must be handled carefully as a collision is unacceptable.

Assume that the first step of Algorithm 1 creates n feasible Dubins paths, denoted $r_i(q) \in \mathbb{R}^2$, where $q \in \mathbb{R}$ is a parameterization variable of the path. For this case, the parameterization variable q can be seen as the path-length or a linear function of time, as it is assumed that all the vehicles are moving with constant speed. Assume further that each agent has a safety radius R_i as it follows the path, where the center of the safety radius conforms with the center of mass of the agent.

Tsourdos et al. [12] proposes to use two safety constraints in order to avoid collision between the agents. The first constraint is called the "*minimum separation distance*", and this constraint is satisfied if the closest distance, $d_{sep,k,m}$ between the paths $r_k(q)$ and $r_m(q)$ satisfy the inequality:

$$d_{sep,k,m} \geq R_k + R_m. \tag{1}$$

This implies that the minimum separation distance is violated if two paths are intersecting. However, an intersection between two paths does not necessarily mean that a collision will arise. As long as the two agents are not arriving at the intersection at the same time, this will not be a problem. Therefore, a second safety constraint is introduced. According to [12] this constraint is called "*non-intersection paths*", but will be named the "*collision constraint*" in this paper. If most of the paths are crossing, it is computationally ineffective to run the "minimum separation distance"-algorithm as it will always fail. As a consequence, only the collision constraint will be used in this work. (There might be some scenarios where it is more computationally efficient to first calculate the minimum separation distance, and then possibly the collision constraint. This will be the case if the minimum separation distance is satisfied, since satisfying this constraint also implies that the collision constraint is satisfied. It will therefore be valid to skip the collision test and in that way optimize the computational effort.) The difference between the "minimum separation distance", $d_{sep,k,m}$ and the "minimum collision distance", $d_{k,m}$ is illustrated in Figure 4.

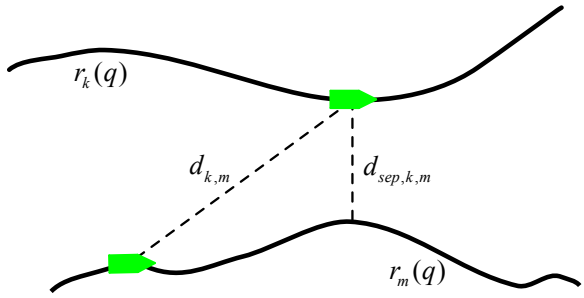

Fig. 4. Graphical figure illustrating the difference between the "minimum separation distance", $d_{sep,k,m}$ and the "minimum collision distance", $d_{k,m}$.

1) Collision definition: A "*collision*" between the two agents k and m, following its respectively paths, $r_k(q)$ and $r_m(q)$, arises if the closest distance, $d_{k,m}$, between the two vehicles satisfy the inequality

$$d_{k,m} \leq R_k + R_m, \tag{2}$$

where R_k is the safety radius of agent k, R_m is the safety radius of agent m and $k, m \in \{1, 2, \cdots, n\}$. The closest distance $d_{k,m}$ between the agents is given by the expression:

$$d_{k,m} = \sqrt{(r_k(q) - r_m(q))^\top (r_k(q) - r_m(q))}, \tag{3}$$

where $q \in [q_0, q_1]$, q_0 is the start value for the parameterization and q_1 is the end value.

The calculation of the "minimum collision distance", can be done in two ways. The easiest way is to let q go from $q_0 \rightarrow q_1$ in (3) and calculate all the corresponding "collision distances". The smallest of all these values will be the "minimum collision distance", $d_{k,m}$. This is unfortunately not a computationally effective method since the "*total expected run time*" [17] is linear ($O(N)$, where N is the number of discretization elements of q). Luckily, it is possible to utilize the fact that the paths consists of only circular arcs and straight segments. Thus, it is possible to reduce the

total running time such that it becomes constant ($O(1)$). Moreover, the solution will also benefit from having an analytic expression and will therefore be exact.

Imagine a scenario as illustrated by Figure 5. The two gray agents represent the present state of the system and it can be seen that $d_{k,m} \leq R_k + R_m$. This is not desired, and a method will therefore be proposed in order to avoid an impact. The gray agents represent the collision, but suppose that it is possible to reconfigure path $r_k(q)$ such that agent m will be located at the black position at the same time instant. Then there will be no collision between the two vehicles. This is the strategy for the path planner in order to make the paths safe. Nevertheless, there are several problems related to this strategy. First of all, it must be decided which path to modify. If the agents are roughly going towards the same goal, it is natural to adjust the path with the agent "lagging behind" (with respect to some fixed point in front of the vehicles). The assumption of vehicles traveling towards approximately the same position is in most cases a reasonable assumption for cooperating missions. For that reason, it is assumed that all the agents are approximately heading towards the same final destination.

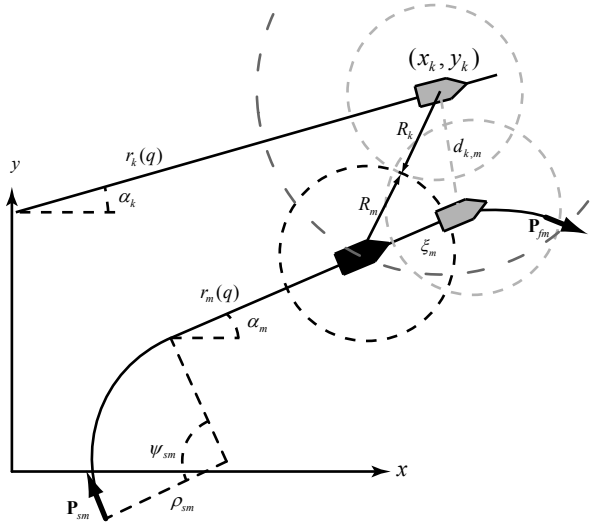

Fig. 5. A snapshot of a possible scenario with two paths $r_k(q)$ and $r_m(q)$. The gray circles and vehicles represent a detected collision. The black circle and vehicle represent a scenario where a collision is barely avoided.

In order to tune path-m, it is preferred that its path length will be increased with the value ξ_m before the collision point. Referring to Figure 5, solving the equation set

$$(x - x_k)^2 + (y - y_k)^2 = (R_k + R_m)^2 \tag{4}$$

$$y = \tan(\alpha_m)x + b \tag{5}$$

for (x, y), where (4) is the equation for the dark gray circle, (5) is the equation for the black straight line, and (α_m, b) is given by analytic expressions from the Dubins path algorithm. The calculation will give two solutions where the solution "behind" agent m is chosen. The preferred point for agent m is then known and the distance ξ_m can easily be calculated. If, on the other hand, agent k is "lagging behind", then ξ_k will be given by the same set of equations, but with agent m as basis and the subscripts k and m switched.

Even though it is easy to calculate the desired adjustment length, it is unfortunately difficult to increase the length of the path such it path becomes collision free. This is because the only way to manipulate the length of the paths is by changing the start radius ρ_{sm}, and finish radius ρ_{fm} (See Figure 3). As the length of the path must be manipulated before the collision, it is in practice only the starting radius that will have significant influence. Hence, the length-increasing method will be controlled by changing the start radius ρ_{sm}.

The length of the start arc will be increased by ξ_m. This will give the modified starting radius, ρ_{sm}^{new}, according to the equation

$$\rho_{sm}^{new} = \frac{\rho_{sm}\psi_{sm} + \xi_m}{\psi_{sm}}, \tag{6}$$

where ψ_{sm} is given by the Dubins path algorithm. The problem that appears now is that the new generated path will be quite different compared to the previous path. This is illustrated in Figure 3 and it is not guaranteed that the newly generated path will be collision free. As a consequence, ρ_{sm}^{new} will only be used as an *initial guess*. If a collision is still present between the paths, it is best to simply increase ρ_{sm} by a fixed step or by a fixed percentage change. This is unfortunately not optimal, but practice shows that the method generally needs only one or two steps for a pair of agents to become collision free. Finally, the pseudo algorithm SAFEPATHS for making the safe paths is shown in Algorithm 2.

Algorithm 2: Generate safe paths.

function SAFEPATHS(P_{si}, P_{fi}, R_i) ▷ $i \in \{1, \cdots, n\}$
  while collision **do**
    for all $i, j \in \{1, \cdots, n\}$ where $i \neq j$ **do**
      if $d_{i,j} \leq R_i + R_j$ **then**
        $k \leftarrow$ index of path "in front".
        $m \leftarrow$ index of path "lagging behind".
        Calculate ρ_{sm}^{new} according to (6).
        $r_m(q) \leftarrow$ New D-path with ρ_{sm}^{new}.
        while $d_{k,m} \leq R_k + R_m$ **do**
          Increase ρ_{sm}.
          $r_m(q) \leftarrow$ New Dubins path with ρ_{sm}.
        end while
      end if
    end for
  end while
  return $r_i(q)$ ▷ Feasible and safe paths.
end function

D. Creating equally long paths

Once the paths are collision free, then the next step is to make them arrive at the same time. Assuming that the speeds of the vehicles are equal, this implies that the lengths of all the paths must be equal. This is done in a straightforward manner.

The method for calculating equally long path consists of two steps. The first step is to identify the longest path,

$|r_k(q)|$, where $k \in \{1, \cdots, n\}$. Then all the remaining paths will be manipulated, through ρ_{si} and ρ_{fi}, such that all path-lengths are equal. This is summarized in Algorithm 3.

Algorithm 3: Generate equally long paths.

function EQUALPATHS(P_{si}, P_{fi}, R_i) ▷ $i \in \{1, \cdots, n\}$
  Find the longest path $|r_k(q)|$, where $k \in \{1, \cdots, n\}$.
  for all $i \neq k$ **do**
    while $|r_i(q)| \neq |r_k(q)|$ **do**
      Increase ρ_{si} and ρ_{fi}.
      $r_i(q) \leftarrow$ New Dubins path with ρ_{si} and ρ_{fi}.
    end while
  end for
  return $r_i(q)$ ▷ Feasible and equally long paths.
end function

When the paths are made equally long, it means that the trajectories are changed and there might be chances for new collisions. The main algorithm must therefore check for collisions again and it may be needed to run through the main algorithm once more. At worst, this can cause an infinite loop, but experience with a small number of vehicles shows that a solution is generally found in one or two steps of the method.

IV. Case Study

In order to test the proposed method, a case study is performed. The algorithm is implemented using Matlab and the results are presented below.

A scenario with three vehicles is simulated ($n = 3$). All vehicles have a safety radius $R_i = 20$ m and a minimum turning radius $\rho_{\min,i} = 10$ m. The initial paths can be seen in Figure 6(a) where the start poses P_{si} and finish poses P_{fi} are illustrated with arrows. This is a quite hard setup that tests the proposed method thoroughly. Most of the time, the main method only needs one or two attempts to satisfy the path requirements. However, with this setup Algorithm 1 needs a total of five rounds before it terminates.

The results of the simulation can be seen in Figure 6–7, where Figure 6 illustrates the paths and Figure 7 shows the path-lengths for each iteration step. The results are satisfactory, and the paths are feasible, safe, and arriving at the same time. Notice especially how "tight" the vehicles are midway during the simulation, as can be seen in Figure 6(b). This illustrates the complexity of the setup, as the black path must be "squeezed" in between the blue and the red.

It is also worth mentioning that the illustrated solution is probably not an optimal solution. For instance, it might be better for one or two of the vehicles to start their paths with a left turn. The vehicle(s) will then travel the "longest initial way", but it may turn out that the total length will be reduced. This is not checked by the presented algorithm; however, only small changes will be necessary to include this. The problem with adding this feature is that the number of solutions will drastically increase, as the Dubins path algorithm gives four solutions for each vehicle. This means that there will be 4^n possible solutions to check, and this will greatly increase the computation time.

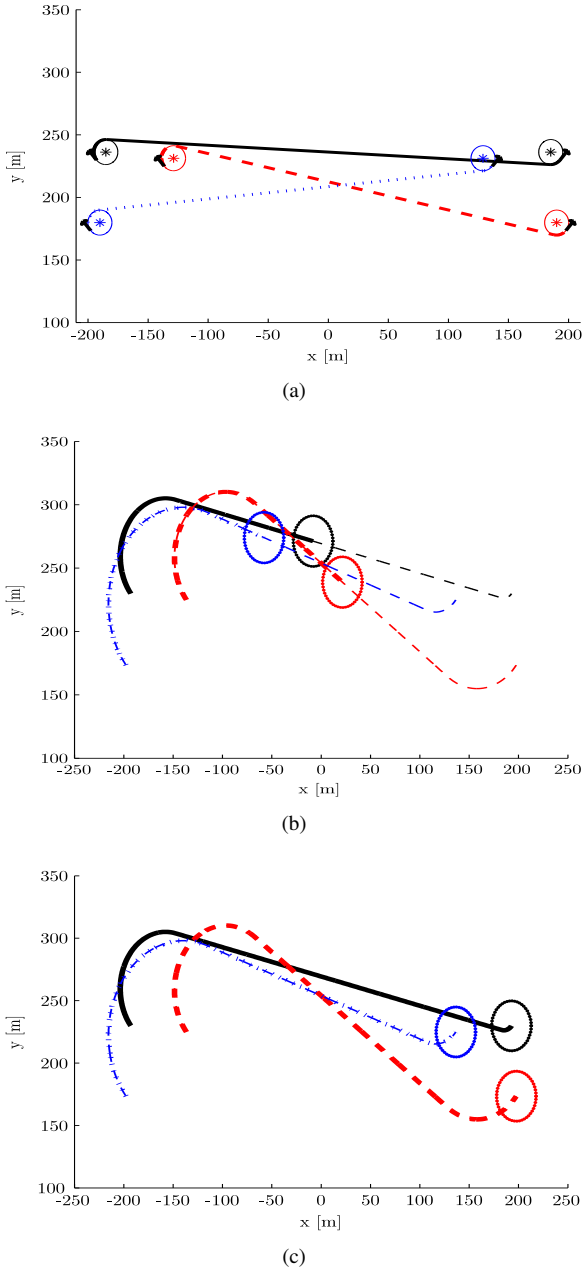

Fig. 6. Simulation of the path algorithm where the initial paths can be seen in (a). A snapshot half way through the simulation is presented in (b), while the final paths are seen in (c). Solid, dashed, and dotted lines correspond to Agent 1–3, respectively.

V. Discussion

As most path planning methods, this method has both advantages and disadvantages. Some of them will be highlighted in this section.

First of all, the main motivation for developing the path planning algorithm is to propose a method that is capable of calculating multiple paths on a system with low CPU capabilities. This is achieved, and experience shows that the setup in Section IV is calculated in about $0,5$ s (calculated in Matlab using a $1,73$ GHz CPU). Furthermore, the authors have not focused much on optimizing the code, and it is

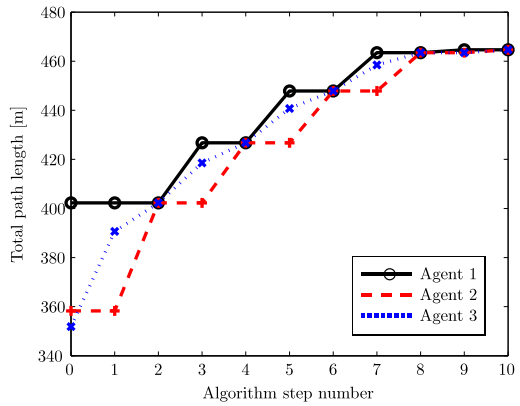

Fig. 7. Total path length after each simulation step. Step 0 is the initial step, while odd and even steps correspond to the result after Algorithm 2 and 3, respectively.

expected that even better performance can be achieved.

Another advantage of the method is the ability to tune the method according to some desired performance. Tuning is mainly achieved by deciding how much the start and finish radii are going to be changed for each step of the algorithm. Choosing large steps usually gives short computation time, while small steps imply higher precision. As a consequence, small steps will in general produce a shorter travel distance, but at the expense of longer computation time.

Unfortunately, there are also some disadvantages related to the method. The main problem is that a solution is not guaranteed. Experience shows that the algorithm in practice finds a solution, but this may not always be the case. There exist some scenarios where the geometry ensures that a solution is possible; however, the method still fails. Although this is quite rare, this is clearly not desired, and research should be undertaken in order to provide guarantees.

Another challenge related to the method is due to the requirement of an initial curvature. In rare cases where the start pose is directly headed towards the finish pose and the two poses are parallel, then the Dubins path algorithm will create a straight line between the poses, with no arc sections. It is therefore impossible to change the length of the path, since the only way to change the path is through the length of the arcs. This can, however, easily be fixed.

Finally, a question to consider is how well the method scales with increasing number of agents. In general, the method works well with many agents as long as each agent has sufficient area available to manipulate its path. However, the requirement of arriving at the same time can in some cases be problematic. This is particularly the case if the initial path lengths are very different, since the algorithm will force the short paths to make large detours. As a result, this may lead to bad behavior, but this is rarely the case when the initial paths are approximately equally long.

VI. Conclusions

In this paper the topic of path planning for multiple autonomous agents has been studied. The algorithm proposed generates Dubins paths that are feasible for a vehicle to follow. Assuming that the speeds of the agents are constant, it is also guaranteed that the paths are collision free and arriving at their target at the same time. In order to implement the method onboard small sized unmanned agents, it has been focused on designing a system that is computationally effective such that the solution can be calculated in a short time without a powerful CPU. Finally the algorithm is tested with a case study that illustrates the method.

References

[1] A. Howard, M. J. Matarić, and S. G. S., "An incremental self-deployment algorithm for mobile sensor networks," *Auton. Robots*, vol. 13, pp. 113–126, September 2002.

[2] M. Jun, *Path Planning for Unmanned Aerial Vehicles in Undcertain and Adversarial Environments*. Springer, 2003, ch. 6, pp. 95–111.

[3] L. F. Bertuccelli and J. P. How, "Search for dynamic targets with uncertain probability maps," in *Proceedings of the American Control Conference, Minneapolis, Minnesota*, june 2006.

[4] J. D. Mot, V. Kulkami, S. Gentry, V. Gavrilets, and E. Feron, "Coordinated path planning for a UAV cluster," in *AINS Symposium, Los Angeles, California*, may 2002.

[5] T. Schouwenaars, J. P. How, and E. Feron, "Multi-vehicle path planning for non-line of sight communication," in *Proceedings of the American Control Conference, Minneapolis, Minnesota*, june 2006.

[6] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[7] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, vol. 2, mar 1985, pp. 500–505.

[8] R. Skjetne, "The maneuvering problem," Ph.D. dissertation, Norwegian University of Science and Technology, 2005.

[9] R. Skjetne, U. Jørgensen, and A. R. Teel, "Line-of-sight path-following along regularly parametrized curves solved as a generic maneuvering problem," in *Proceedings of the 50th IEEE Conference on Decision and Control.*, vol. 50th. Orlando, USA: IEEE, Dec. 2011.

[10] T. W. McLain and R. W. Beard, "Trajectory planning for coordinated rendezvous of unmanned air vehicles," in *proceedings of AIAA Guidance, Navigation and Control Conference*, 2000.

[11] T. W. McLain, P. R. Chandler, S. Rasmussen, and M. Pachter, "Cooperative control of UAV rendezvous," in *Proceedings of the 2001 American Control Conference*, vol. 3, 2001, pp. 2309–2314.

[12] A. Tsourdos, B. White, and M. Shanmugavel, *Cooperative Path Planning of Unmanned Aerial Vehicles*. John Wiley & Sons, Ltd, 2011.

[13] J.-P. Laumond, "Feasible trajectories for mobile robots with kinematic and environment constraints," in *Intelligent Autonomous Systems, An International Conference*. North-Holland Publishing Co., 1987, pp. 346–354.

[14] M. Shanmugavel, A. Tsourdos, R. Żbikowski, and B. A. White, "Path planning of multiple UAVs with clothoid curves," in *Proceedings of the 17th IFAC Symp. on Automatic Control in Aerospace*, 2007.

[15] R. T. Farouki and T. Sakkalis, "Pythagorean hodographs," *IBM Journal of Research and Development*, vol. 34, no. 5, pp. 736–752, sept. 1990.

[16] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2002.